APX publishes programs in these categories:

MATHLIB

SYSTEMS/TELECOMMUNICATIONS

Add floating point capability to your DEEP BLUE C COMPILER (for programmers familiar with DEEP BLUE C)

by Frank Paris

Requires: DEEP BLUE C COMPLILER (APX-20166)
Full Screen editor
Diskette version (1): (APX–20231)
ATARI 810 or 1050 Disk Drive
48K RAM

Edition A

CONSUMER-WRITTEN PROGRAMS FOR

ATARI® HOME COMPUTERS

APX
ATARI Program Exchange

Printed in U.S.A.

MATHLIB

SYSTEMS/TELECOMMUNICATIONS

Add floating point capability to your DEEP BLUE C COMPILER
(for programmers familiar with DEEP BLUE C)

by Frank Paris

Requires: DEEP BLUE C COMPLILER (APX-20166)
Full Screen editor
Diskette version (1): (APX–20231)
ATARI 810 or 1050 Disk Drive
48K RAM

Edition A

MATHLIB

by

Frank Paris

Program and manual contents ©1983 Frank Paris

Copyright notice. On receipt of this computer program and associated documentation (the software), the author grants you a nonexclusive license to execute the enclosed software. This software is copyrighted. You are prohibited from reproducing, translating, or distributing this software in any unauthorized manner.

Distributed By

The ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

To request an APX Product Catalog, write to the address above, or call toll-free:

800/538-1862 (outside California)
800/672-1850 (within California)

Or call our Sales number, 408/727-5603

Trademarks of Atari

ATARI is a registered trademark of Atari, Inc. The following are trademarks of Atari, Inc: 400, 410, 800, 810, 820, 822, 825, 830, 850, 1200XL.

Limited Warranty on Media and Hardware Accessories. Atari, Inc. ("Atari") warrants to the original consumer purchaser that the media on which APX Computer Programs are recorded and any hardware accessories sold by APX shall be free from defects in material or workmanship for a period of thirty (30) days from the date of purchase. If you discover such a defect within the 30-day period, call APX for a return authorization number, and then return the product to APX along with proof of purchase date. We will repair or replace the product at our option. If you ship an APX product for in-warranty service, we suggest you package it securely with the problem indicated in writing and insure it for value, as Atari assumes no liability for loss or damage incurred during shipment.

This warranty shall not apply if the APX product has been damaged by accident, unreasonable use, use with any non-ATARI products, unauthorized service, or by other causes unrelated to defective materials or workmanship.

Any applicable implied warranties, including warranties of merchantability and fitness for a particular purpose, are also limited to thirty (30) days from the date of purchase. Consequential or incidental damages resulting from a breach of any applicable express or implied warranties are hereby excluded.

The provisions of the foregoing warranty are valid in the U.S. only. This warranty gives you specific legal rights and you may also have other rights which vary from state to state. Some states do not allow limitations on how long an implied warranty lasts, and/or do not allow the exclusion of incidental or consequential damages, so the above limitations and exclusions may not apply to you.

Disclaimer of Warranty on APX Computer Programs. Most APX Computer Programs have been written by people not employed by Atari. The programs we select for APX offer something of value that we want to make available to ATARI Home Computer owners. In order to economically offer these programs to the widest number of people, APX Computer Programs are not rigorously tested by Atari and are sold on an "as is" basis without warranty of any kind. Any statements concerning the capabilities or utility of APX Computer Programs are not to be construed as express or implied warranties.

Atari shall have no liability or responsibility to the original consumer purchaser or any other person or entity with respect to any claim, loss, liability, or damage caused or alleged to be caused directly or indirectly by APX Computer Programs. This disclaimer includes, but is not limited to, any interruption of services, loss of business or anticipatory profits, and/or incidental or consequential damages resulting from the purchase, use, or operation of APX Computer Programs.

• Some states do not allow the limitation or exclusion of implied warranties or of incidental or consequential damages, so the above limitations or exclusions concerning APX Computer Programs may not apply to you.

Table of contents

INTRODUCTION...1

Overview...1
Required accessories...1
Optional accessories...1
Contacting the author...1

WHAT DO YOU GET WITH MATHLIB?...2

Summary of MATHLIB commands...2
Required files not supplied with MATHLIB...3
Files included with MATHLIB...3

USING MATHLIB...5

Initializing MATHLIB...5
Declaring floating point variables...5
Creating floating point constants...6
Printing a floating point number...7
Integer/floating point conversions...7

MATHLIB FUNCTION SPECIFICATIONS...9

Purpose...9
Function call...9
Input parameters...9
Output parameters...9
Description...9
Functions used...9
Example...10

FUNCTIONS IN 'MATHLIB.CCC'...11

Initialize MATHLIB (c_iml)...12
ATASCII to floating point conversion (c_afp)...13
Floating point to ATASCII conversion (c_fasc)...14
Unsigned integer to floating point conversion (c_ifp)...15
Signed integer to floating point conversion (c_sifp)...16
Floating point to unsigned integer conversion (c_fpi)...17
Floating point to signed integer conversion (c_fpsi)...18
Floating point addition (c_fAdd)...19
Floating point subtraction (c_fsub)...20
Floating point multiplication (c_fmul)...21
Floating point division (c_fdiv)...22
Floating point logarithm (c_log and c_log10)...23
Floating point antilogarithm (c_alog and c_alog10)...24
Floating point exponentiation (c_exp)...25

Floating point square root (c_sqrt)...26
Return integer portion of a number (c_int)...27
Return fractional portion of a number (c_frac)...28
Compare two floating point numbers (c_cmp)...29
Get absolute value of number (c_abs)...30
Change sign of floating point number (c_chs)...31
Set floating point number to zero (c_zero)...32
Move floating point number (c_move)...33

TRIGONOMETRIC FUNCTIONS (TRIG.CCC)...34

Initialize trigonometric functions (c_itrig)...35
Set radians to degrees (c_rad)...36
Convert radians to degrees (c_rd)...37
Convert degrees to radians (c_dr)...38
Degrees, minutes, seconds to decimal degrees (c_dmsd)...39
Decimal degrees to degrees, minutes, seconds (c_ddms)...40
Compute sine of an angle (c_sin)...41
Compute cosine of an angle (c_cos)...42
Compute tangent of an angle (c_tan)...43
Compute arctangent (c_atan)...44

Introduction

OVERVIEW

MATHLIB is a library of C-Language functions that allows you to do floating point calculations with the ATARI Deep Blue C Compiler (APX-20166). The Deep Blue C Compiler provides a wonderful programming language for the ATARI computer, infinitely more interesting and enjoyable and several times faster than BASIC. Unfortunately, it doesn't support floating point data types. Among other things, this makes Deep Blue C, by itself, next to impossible to use in advanced graphics applications, the strong suit of the ATARI Home Computer. This is because advanced graphics make extensive use of floating point numbers and trigonometric functions. MATHLIB fills this gap in the Deep Blue C Compiler.

Using MATHLIB to perform floating point calculations isn't as concise as doing integer arithmetic in C, but it does provide a full range of floating point mathematical functions. A demonstration program is included to illustrate how to implement turtle graphics (with window clipping) using Deep Blue C with MATHLIB.

This manual assumes that you're already familiar with the C programming language in general and ATARI Deep Blue C in particular.

REQUIRED ACCESSORIES

DEEP BLUE C COMPILER (APX-20166)
Full screen editor

OPTIONAL ACCESSORIES

ATARI MACRO Assembler

CONTACTING THE AUTHOR

Users wishing to contact the author may write to him at:

6855 SW Murray Blvd.
Beaverton, OR 97005

What do you get with MATHLIB?

MATHLIB operates on standard ATARI floating point numbers and provides access to the functions in the ATARI Operating System Floating Point ROM. MATHLIB provides many extensions of the ROM capabilities, including trigonometric functions. All together, MATHLIB provides you with 32 new math functions for your Deep Blue C Compiler.

SUMMARY OF MATHLIB FUNCTIONS

The following is a summary of the functions provided by MATHLIB:

- ATASCII to floating point and floating point to ATASCII conversions;
- signed and unsigned integer to floating point and floating point to integer conversions;
- addition, subtraction, multiplication, and division;
- natural and base 10 logarithms and exponentiation;
- square root;
- sine, cosine, tangent, and arctangent trigonometric functions in either radians or degrees;
- radian to degree and degree to radian conversions;
- decimal degrees to degrees, minutes, and seconds, and vice versa;
- integer or fractional portion of a floating point number;
- comparison of two floating point numbers;
- absolute value of a floating point number;
- change sign of a floating point number;
- set floating point number to zero
- move floating point number

REQUIRED FILES NOT SUPPLIED WITH MATHLIB

MATHLIB assumes that you already have the ATARI Deep Blue C Compiler. The following files from Deep Blue C are the minimum required for using MATHLIB (they're not included with MATHLIB):

- CC.COM -- Deep Blue C Compiler
- CLINK.COM -- Deep Blue C Linker
- DBC.OBJ -- C run-time library
- AIO.CCC -- object for I/O functions

In addition, to construct the turtle graphics demonstration program included with MATHLIB, you must have the following Deep Blue C files (they also aren't included):

GRAPHICS.CCC -- object for graphic and game I/O

PRINTF.CCC -- object for formatted output

FILES INCLUDED WITH MATHLIB

The following files are included on the MATHLIB diskette:

- MATHLIB.C -- source for the non-trigonometric functions of MATHLIB; this includes all functions except the trigonometric functions.
- MATHLIB.CCC -- object for MATHLIB.C. This must always be included in your Deep Blue C link file.
- TRIG.C -- source for the trigonometric functions of MATHLIB.
- TRIG.CCC -- object for TRIG.C. This needs to be included in your Deep Blue C link file only if you're using the MATHLIB trigonometric functions.
- MATHLIB.OBJ -- object for the assembly language interface to the ATARI floating point ROM. This must always be included in your Deep Blue C link file. MATHLIB.OBJ is fixed at ATARI RAM hexadecimal

locations 2DC0 to 2FFF. If you use MATHLIB with your own assembly language routines, you must ensure that they don't use the same locations as MATHLIB.OBJ. Alternatively, if you have the ATARI MACRO Assembler, you can use the next file to reassemble MATHLIB.OBJ at a new starting location. In that case, you will also have to change the entry points to MATHLIB.OBJ in the MATHLIB.CCC file and recompile MATHLIB.CCC.

- MATHLIB.ASM -- Assembly language source for MATHLIB.OBJ. (Requires ATARI MACRO Assembler CX8121 to assemble).

- TURTLE.C -- source for the mainline MATHLIB demonstration program. Includes mathematical functions to execute the basic turtle graphic movements of direction and distance.

- TURTLE.CCC -- object for TURTLE.C.

- CLIPPER.C -- source for the Cohen-Sutherland Clipping Algorithm adapted to the ATARI Deep Blue C Language. Contains functions called by TURTLE.

- CLIPPER.CCC -- object for CLIPPER.C.

- TURTLE.LNK -- Deep Blue C link file to generate the demonstration program.

- TURTLE.COM -- Executable load module of the turtle graphics demonstration program

Note. All MATHLIB function names start with 'c_' to avoid any conflicts with names in your own programs.

INITIALIZING MATHLIB

Before using MATHLIB, you must call the MATHLIB function, 'c_iml'. If you use the trigonometric functions, you must also call 'c_itrig'. These functions initialize the constants and variables that MATHLIB must use to perform its functions. Neither of these routines has any parameters and so they're called as follows:

```
c_iml();
c_itrig();
```

Note that 'c_itrig' is called only if the trigonometric functions are used and TRIG.CCC is linked into your load module. 'c_iml' must be called before 'c_itrig'.

DECLARING FLOATING POINT VARIABLES

MATHLIB uses standard ATARI floating point numbers. You don't have to understand the format of ATARI floating point numbers to use MATHLIB, and it's beyond the scope of this manual to explain it. But if you're curious refer to the APX publication, De Re ATARI (APX-90008), pp. 8-45 and 8-46. The only thing you have to know is that ATARI floating point numbers each occupy 6 bytes of ATARI RAM.

MATHLIB uses C language character arrays to hold floating point numbers. To declare a variable that will hold a floating point number, code the following (assuming the name of your variable is 'fpvar'):

```
char fpvar[6];
```

All MATHLIB floating point variables must be declared in that manner. On the other hand, all function arguments referencing floating point numbers in MATHLIB functions are character pointers. This means that when you reference a floating point number in a MATHLIB function, you simply use the name of the variable without including the subscript. For example, suppose you want to add two floating point numbers, 'fa' and 'fb', and you

want the results stored in 'fb'. The variables must have been declared as follows:

```
char fa[6], fb[6];
```

The MATHLIB function that adds two floating point numbers is c_fadd. It accepts three arguments: the two numbers to be added together, followed by the result. Thus to add the two numbers, you would code:

```
c_fadd (fa, fb, fb);
```

This adds 'fa' to 'fb' and stores the results in 'fb'. Notice that you specify the name of the variable only, omitting the subscripts.

CREATING FLOATING POINT CONSTANTS

Deep Blue C does not support the standard C language "float" data type. Therefore, it's impossible to declare a floating point constant in that language. For this reason, constants can never be explicitly passed as arguments to a MATHLIB function. First create the floating point constant by the appropriate MATHLIB routine, which moves it to a six byte character array you've declared in your program.

Floating point constants start out as ATASCII character strings, which are converted in a single step to the standard ATARI floating point representation. This is done with the 'c_afp' MATHLIB function, which stands for "ATASCII to Floating Point". 'c_afp' uses two character pointers as arguments. The first points to the ATASCII character string that represents the constant, and the second points to the six byte character array that will receive the floating point representation of the constant.

For example, suppose you wish to create a floating point representation of the constant, pi (3.14159265) and store it in character array 'fppi'. You'd code the following:

```
char fppi[6], *pntr;
pntr = "3.14159265";
c_afp (pntr, fppi);
```

In this example, 'pntr' is set to point to the ATASCII character string and 'c_afp' is used to convert that string to its ATARI floating point representation, which is stored in the six byte

character array, 'fppi'. 'fppi' may now be used in function calls to pass the constant, pi, to other routines.

PRINTING A FLOATING POINT NUMBER

You can't print an ATARI floating point number directly. You must convert it to an ATASCII character string with the MATHLIB function, 'c_fasc', which stands for "Floating Point to ATASCII." This function accepts two character pointers as arguments. The first points to the floating point number, and the second points to a character array that will receive the ATASCII representation of the number. The character array holding the converted number can then be used in a 'printf' command to print the floating point number.

Before giving an example, the size of the array for the ATASCII representation of the floating point number must be considered. ATARI floating point numbers provide up to ten digits of precision. In addition, the number may be prefaced with a minus sign. The number may also include a decimal point or a signed two digit exponent of the form, 'E-xx' where 'xx' is the exponent. Finally, the ATASCII floating point number will be followed by a null character (the C language standard for character strings). This all adds up to 17 characters. Thus, the character array to hold the converted floating point number must be declared 17 characters long.

Now for the example. Suppose you have a floating point number in a 6 character array, 'fpnbr', and you want to print it, using 'printf'. You could do it with the following code:

```
char fpnbr[6]; /* floating point number */
char arfpnbr[17]; /* ATASCII representation of
     the f.p. number */
c_fasc (fpnbr, arfpnbr); /* convert f.p. to
     ATASCII */
printf ("%s", arfpnbr); /* print ATASCII
     representation */
```

INTEGER/FLOATING POINT CONVERSIONS

MATHLIB provides functions for converting back and forth between integers and floating point numbers. MATHLIB distinguishes between two kinds of integers: unsigned 16 bit numbers in the range 0 to 65,535 and signed numbers in the range -32,768 to +32,767. Unsigned numbers are of limited value, since they aren't supported by Deep Blue C (Deep Blue C will treat an

unsigned number greater than 32,767 as a negative number. However, if you know your floating point number isn't negative, it's slightly faster to use the unsigned MATHLIB functions, since it's unsigned integers that the ATARI floating point ROM deals with directly. Signed integers require extra processing on the part of MATHLIB.

If you call the unsigned floating point to integer MATHLIB function and the floating point number is greater than 65,535, you'll receive error status back from the function. Likewise, if you call the signed floating point to integer MATHLIB function and the floating point number is greater than 32,767, you will receive error status back. Remember, if you attempt to use an unsigned integer greater than 32,767 with Deep Blue C, it will treat it as a negative number.

The four conversion routines are as follows. (Details may be found in a later section of this manual.)

- c_ifp -- unsigned integer to floating point
- c_fpi -- unsigned floating point to integer
- c_sifp -- signed integer to floating point
- c_sfpi -- signed floating point to integer

MATHLIB function specifications

The following sections of this manual contain descriptions of each function within MATHLIB. Each description contains the following seven headings:

PURPOSE

This is a one or two sentence description of the purpose of the function.

FUNCTION CALL

This is an example of how the function is called. It shows all the parameters you should include when you call the function. An exception is the 'status' return value of the function. The return value is usually an indication of whether the operation was carried out successfully or not. The main reason why it may have failed is an out of range condition: the result of the function may be out of the range of values that a standard ATARI floating point value can take. This range is 10**-98 to 10**+98. If you know that the answer must be within this range, you don't have to include the 'status' return value in your call to the function.

INPUT PARAMETERS

This describes all the parameters input by the calling routine to the function. The first line of each description shows the name of the parameter as used in the FUNCTION CALL, as well as the data type of the parameter. Each is followed by a description of the parameter.

OUTPUT PARAMETERS

This describes all the parameters that are output by the function to the calling function.

DESCRIPTION

This section provides a description of what the function actually does in terms of the input and output parameters.

FUNCTIONS USED

This section lists the functions used to implement the function being described. These functions may be other MATHLIB functions or functions from the AIO.CCC Deep Blue C library. If nothing but standard features of the Deep Blue C language itself are used, "None" appears under this heading.

EXAMPLE

This optional section gives an example of what the function does. If it's obvious from the description of the function what it does, an example isn't given. The examples usually start with ATASCII character strings representing floating point numbers, convert them to ATARI floating point, perform the function, convert them back to ATASCII and then print the result with 'printf'. The printed results follow the 'printf' statement in italics.

Within the various sections, if a parameter name is used in descriptive text, it appears in single quotation marks. If the parameter is a pointer and the value pointed to is intended, an asterisk preceeds the variable, within the single quotation marks.

Functions in 'MATHLIB.CCC'

The functions described below, in the file, MATHLIB.CCC, are the non-trigonometric functions of MATHLIB. The following is a complete list of these functions in the order they're specified in the following pages:.

c_iml:	Initialize MATHLIB.CCC
c_afp:	ATASCII to Floating Point Conversion
c_fasc:	Floating Point to ATASCII Conversion
c_ifp:	Unsigned Integer to Floating Point Conversion
c_sifp:	Signed Integer to Floating Point Conversion
c_fpi:	Floating Point to Unsigned Integer Conversion
c_fpsi:	Floating Point to Signed Integer Conversion
c_fadd:	Floating Point Addition
c_fsub:	Floating Point Subtraction
c_fmul:	Floating Point Multiplication
c_fdiv:	Floating Point Division
c_log:	Floating Point Natural Logarithm
c_log10:	Floating Point Common Logarithm
c_alog:	Floating Point Natural Antilogarithm
c_alog10:	Floating Point Common Antilogarithm
c_exp:	Floating Point Exponentiation
c_sqrt:	Floating Point Square Root
c_int:	Return Integer Portion of Floating Point Number
c_frac:	Return Fractional Portion of Floating Point Number
c_cmp:	Compare Two Floating Point Numbers
c_abs:	Get Absolute Value of Floating Point Number
c_chs:	Change Sign of Floating Point Number
c_zero:	Set Floating Point Number to Zero
c_move:	Move Floating Point Number

INITIALIZE MATHLIB

PURPOSE

To initialize the constants and variables required by MATHLIB to perform its functions.

FUNCTION CALL

```
c_iml();
```

INPUT PARAMETERS

None

OUTPUT PARAMETERS

None

DESCRIPTION

This function must be called once before any MATHLIB routine is executed.

FUNCTIONS USED

c_afp

ATASCII TO FLOATING POINT CONVERSION

PURPOSE

To convert an ATASCII character string representation of a floating point number to ATARI floating point format.

FUNCTION CALL

```
status = c_afp (acs, fpn);
```

INPUT PARAMETERS

acs char array
pointer to a character string containing the ATASCII representation of a floating point number.

OUTPUT PARAMETERS

fpn char array
pointer to a six-byte character array that will receive the standard ATARI floating Point number corresponding to the ATASCII input number.

status integer scalar
return status:
0 = ATASCII number converted correctly.
-1 = the first byte of ATASCII number is invalid.

DESCRIPTION

This function takes bytes from '*acs' until it encounters a byte that can't be part of the number. The bytes scanned to that point are then converted to a floating point number, which is stored in array 'fpn', which must be six characters long. If the first byte encountered in '*acs' is invalid, 'status' is set to -1. Otherwise it's set to 0.

FUNCTIONS USED

This function calls the ATARI floating point ROM directly.

EXAMPLE

```
char pntr, fpn[6];
pntr = "56.789";
c_afp (pntr, fpn);
```

FLOATING POINT TO ATASCII CONVERSION

PURPOSE

To convert a standard ATARI floating point number to a standard C-Language character string, suitable for printing.

FUNCTION CALL

```
c_fasc (fpn, acs);
```

INPUT PARAMETERS

fpn char array
pointer to six byte character array that contains a floating point number in standard ATARI format.

OUTPUT PARAMETERS

acs char array
pointer to 17 byte character array that will contain the printable ATASCII representation of the floating point number.

DESCRIPTION

This function converts the floating point number in 'fpn' to a printable form (ATASCII) in the character array, 'acs', which must be at least 17 bytes long. No error conditions are detected by this function.

FUNCTIONS USED

This function calls the ATARI floating point ROM directly.

EXAMPLE

```
char pntr, fpn[6], output[17];
pntr = "56.789";
c_afp (pntr, fpn);
c_fasc (fpn, output);
printf ("%s", output);
56.789
```

UNSIGNED INTEGER TO FLOATING POINT CONVERSION

PURPOSE

To convert an unsigned integer (0 to 65,535) to a standard ATARI floating point number.

FUNCTION CALL

```
c_ifp (usint, fpn);
```

INPUT PARAMETERS

usint integer scalar
unsigned integer (0 to 65,535) to be converted to floating point

fpn char array
pointer to six byte character array to receive converted floating point number.

DESCRIPTION

This function converts the unsigned integer in 'usint' to a standard ATARI floating point number and stores the results in the six byte character array pointed to by 'fpn'. This function detects no error conditions. Note that Deep Blue C does not support unsigned integers. All unsigned integers greater than 32,767 are treated as negative integers by Deep Blue C.

FUNCTIONS USED

This function calls the ATARI floating point ROM directly.

EXAMPLE

```
int integer;
char fpn[6], output[17];
integer = -5000;
c_ifp (integer, fpn);
c_fasc (fpn, output);
printf ("%s", output);
60536
```

SIGNED INTEGER TO FLOATING POINT CONVERSION

PURPOSE

To convert a signed integer (-32768 to +32,767) to a standard ATARI floating point number.

FUNCTION CALL

```
c_sifp (sint, fpn);
```

INPUT PARAMETERS

sint integer scalar
signed integer (-32,768 to +32,767) to be converted to floating point

OUTPUT PARAMETERS

fpn char array
pointer to six byte character array to receive converted floating point number.

DESCRIPTION

This function converts the signed integer in 'sint' to a standard ATARI floating point number and stores the results in the six byte character array pointed to by 'fpn'. This function detects no error conditions.

FUNCTIONS USED

This function calls 'c_ifp' to implement its functionality. Thus, it's slightly less efficient to use this function for positive integers than 'c_ifp' directly.

EXAMPLE

```
int integer;
char fpn[6], output[17];
integer = -5000;
c_sifp (integer, fpn);
c_fasc (fpn, output);
printf ("%s", output);
-5000
```

FLOATING POINT TO UNSIGNED INTEGER CONVERSION

PURPOSE

To convert a standard ATARI floating point number to an unsigned integer (0 to 65,635).

FUNCTION CALL

```
status = c_fpi (fpn, &usint);
```

INPUT PARAMETERS

fpn char array

pointer to 6 byte array containing a standard ATARI floating point number to be converted to an unsigned integer.

OUTPUT PARAMETERS

usint integer scalar

integer variable to receive the converted unsigned integer.

status integer scalar

returned status;
0 = floating point number converted successfully.
-1 = floating point number is >= 65,535.5;
no conversion performed.
-2 = floating point number is negative;
no conversion performed.

DESCRIPTION

This function converts the standard ATARI floating point number in 'fpn' to an unsigned integer. If the floating point number is negative, -2 is returned as status; no conversion is performed. If the floating point number is greater than or equal to 65,535.5, -1 is returned as status; no conversion is performed. This function performs true rounding, not truncation, during conversion.

FUNCTIONS USED

This function calls ATARI floating point ROM directly.

EXAMPLE

```
char pntr, fpn[6];
int integer;
pntr = "60000";
c_afp (pntr, fpn);
c_fpi (fpn, &integer);
printf ("%d", integer);
-5536
```

FLOATING POINT TO SIGNED INTEGER CONVERSION

PURPOSE

To convert a standard ATARI floating point number to a signed integer (-32,768 to +32,767).

FUNCTION CALL

```
status = c_fpsi (fpn, &sint);
```

INPUT PARAMETERS

fpn char array
pointer to six byte character array containing the floating point number to be converted.

OUTPUT PARAMETERS

sint integer scalar
integer variable to receive the converted signed integer.

status return status:
0 = floating point number converted successfully.
-1 = absolute value of floating point number > 32,767.5.

DESCRIPTION

This function converts the standard floating point number in 'fpn' to a signed integer. If the floating point number is greater than or equal to 32,767.5, -1 is returned as status and no conversion is performed.

FUNCTIONS USED

This function calls 'c_fpi' to implement its functionality. Thus, it's slightly less efficient to use this function for positive integers than 'c_fpi' directly.

EXAMPLE

```
char pntr, fpn[6];
int integer;
pntr = "60000";
c_afp (pntr, fpn);
c_fpsi (fpn, &integer);
printf ("%d", integer);
-5536
```

FLOATING POINT ADDITION

PURPOSE

To add two standard ATARI floating point numbers.

FUNCTION CALL

```
status = c_fadd (fpn1, fpn2, fpsum);
```

INPUT PARAMETERS

fpn1 char array
pointer to a six byte character array containing the first floating point number.

fpn2 char array
pointer to a six byte character array containing the second floating point number.

OUTPUT PARAMETERS

fpsum char array
pointer to a six byte character array that will receive the sum of the first two floating point numbers.

status integer scalar
return status;
0 = addition performed correctly; -1 = out of range result.

DESCRIPTION

This function adds '*fpn1' to '*fpn2' and stores the result at 'fpsum'. If it's outside the range of ATARI floating point number format, -1 is returned as status. If the operation completes successfully, 0 is returned as status. 'fpn1' and 'fpn2' may be the same pointer and 'fpsum' may be the same pointer as 'fpn1' or 'fpn2'.

FUNCTIONS USED

This function calls ATARI floating point ROM directly.

EXAMPLE

```
char *pntr, fp1[6], fp2[6], output[17];
pntr = "321.12";
c_afp (pntr, fp1);
pntr = "21.123";
c_afp (pntr, fp2);
c_fadd (fp1, fp2, fp2);
c_fasc (fp2, output);
printf ("%s", output);
342.243
```

FLOATING POINT SUBTRACTION

PURPOSE

To subtract one standard ATARI floating point number from another.

FUNCTION CALL

```
status = c_fsub (minuend, subtrahend, difference);
```

INPUT PARAMETERS

minuend	char	array

pointer to a six character array containing the minuend of the subtraction operation.

subtrahend	char	array

pointer to a six character array containing the subtrahend of the subtraction.

OUTPUT PARAMETERS

difference	char	array

pointer to a 6 character array that will contain the difference between the minuend and the subtrahend.

status	integer	scalar

return status:
0 = subtraction performed correctly;
-1 = out of range result.

DESCRIPTION

This function subtracts '*subtrahend' from '*minuend' and stores the result at 'difference'. If it's outside the range of ATARI floating point numbers, -1 is returned in 'status'. Otherwise 0 is returned, showing successful computation. 'difference' may be the same pointer as 'minuend' or 'subtrahend'.

FUNCTIONS USED

This function calls ATARI floating point ROM directly.

EXAMPLE

```
char *pntr, fp1[6], fp2[6], output[17];
pntr = "321.12";
c_afp (pntr, fp1);
pntr = "21.123";
c_afp (pntr, fp2);
c_fsub (fp1, fp2, fp2);
c_fasc (fp2, output);
printf ("%s", output);
299.997
```

FLOATING POINT MULTIPLICATION

PURPOSE

To multiply two standard ATARI floating point numbers together.

FUNCTION CALL

```
status = c_fmul (multiplicand, multiplier, product);
```

INPUT PARAMETERS

multiplicand char array
pointer to a 6 character array containing the multiplicand of the multiplication operation.

multiplier char array
pointer to a 6 character array containing the multiplier of the multiplication operation.

OUTPUT PARAMETERS

product char array
pointer to a six character array that will contain the product of the multiplication.

status integer scalar
return status:
0 = multiplication performed correctly.
-1 = out of range result.

DESCRIPTION

This function multiplies '*multiplicand' by '*multiplier' and stores the result at 'product'. If the result is outside the range of standard ATARI floating point numbers, -1 is returned in 'status'. Otherwise 0 is returned, showing a successful computation. 'multiplicand' and 'multiplier' may be the same pointer and 'product' may be the same pointer as 'multiplicand' or 'multiplier'.

FUNCTIONS USED

This function calls ATARI floating point ROM directly.

EXAMPLE

```
char *pntr, fp1[6], fp2[6], output[17];
pntr = "321.12";
c_afp (pntr, fp1);
pntr = "21.123";
c_afp (pntr, fp2);
c_fmul (fp1, fp2, fp2);
c_fasc (fp2, output);
printf ("%s", output);
6783.01776
```

FLOATING POINT DIVISION

PURPOSE

To divide one standard ATARI floating point number by another.

FUNCTION CALL

```
status = c_fdiv (dividend, divisor, result);
```

INPUT PARAMETERS

dividend char array
pointer to a six byte character array containing the dividend of the divide operation.

divisor char array
pointer to a six byte character array containing the divisor of the divide operation.

OUTPUT PARAMETERS

result pointer to a six character array that will contain the result of the division operation.

status integer scalar
return status:
0 = division was successful
-1 = out of range result or divisor is zero.

DESCRIPTION

'*divisor' is divided into '*dividend' and the result is stored at 'result'. If the result is out of the range of standard ATARI floating point numbers or the divisor is zero, -1 is returned as status. Otherwise, 0 is returned. 'result' may be the same pointer as 'dividend' or 'divisor'.

FUNCTIONS USED

This function directly calls ATARI floating point ROM.

EXAMPLE

```
char *pntr, fp1[6], fp2[6], output[17];
pntr = "321.12";
c_afp (pntr, fp1);
pntr = "21.123";
c_afp (pntr, fp2);
c_fdiv (fp1, fp2, fp2);
c_fasc (fp2, output);
printf ("%s", output);
15.202386
```

FLOATING POINT LOGARITHM

PURPOSE

To find the logarithm of a standard ATARI floating point number.

FUNCTION CALL

```
status = c_log (nbr, log);
status = c_log10 (nbr, log);
```

INPUT PARAMETERS

nbr	char	array

pointer to a six byte character array containing a standard ATARI floating point number whose logarithm is desired.

OUTPUT PARAMETERS

log	char	array

pointer to a six byte character array that will receive the logarithm of '*nbr'.

status	integer	scalar

return status:
0 = logarithm successfully computed.
-1 = negative number or overflow.

DESCRIPTION

'c_log' takes the natural logarithm (base e) and 'c_log10' takes the common logarithm (base 10). If '*nbr' is negative or an overflow results, 'result' is set to -1. Otherwise it is set to 0. 'nbr' and 'log' can be the same pointer.

FUNCTIONS USED

Both functions directly call the ATARI floating point ROM.

EXAMPLE

```
char *pntr, nbr[6], log[6], answer[17];
pntr = "256.512";
c_afp (pntr, nbr);
c_log (nbr, log);
c_fasc (log, answer);
printf ("%s", answer);
5.5471754
```

FLOATING POINT ANTILOGARITHM

PURPOSE

To find the antilogarithm of a standard ATARI floating point number.

FUNCTION CALL

```
status = c_alog (nbr, antilog);
status = c_alog10 (nbr, antilog);
```

INPUT PARAMETERS

nbr char array

pointer to a six byte character array containing a standard ATARI floating point number whose antilog is desired.

OUTPUT PARAMETERS

antilog char array

pointer to a six byte character array that will receive the antilogarithm of '*nbr'.

status integer scalar

return status:
0 = antilog taken successfully
-1 = overflow

DESCRIPTION

'c_alog' takes the natural antilog and 'c_alog10' takes the common antilog. The natural log is e (2.7182818) raised to the power '*nbr'. The common antilog is 10 raised to the power '*nbr'. If an overflow results, -1 is returned as status. Otherwise 0 is returned. 'nbr' and 'antilog' can be the same pointer.

FUNCTIONS USED

Both functions directly call the ATARI floating point ROM.

EXAMPLE

```
char *pntr, nbr[6], log[6], answer[17];
pntr = "5.5471754";
c_afp (pntr, nbr);
c_alog (nbr, log);
c_fasc (log, answer);
printf ("%s", answer);
256.512
```

FLOATING POINT EXPONENTIATION

PURPOSE

To raise a standard ATARI floating point number to the power of another one.

FUNCTION CALL

```
status = c_exp (base, exponent, result);
```

INPUT PARAMETERS

base	char	array

pointer to a 6 byte character array containing a standard ATARI floating point number to be raised to a power.

exponent	char	array

pointer to a six byte character array containing a standard ATARI floating point number to be used as the exponent of the number at 'base'.

OUTPUT PARAMETERS

result	char	array

pointer to a 6 byte character array that will be set to the number resulting from raising '*base' to the power '*exponent'.

status	integer	scalar

return status; 0 = operation completed successfully; -1 = out of range

DESCRIPTION

The number at 'base' is raised to the power at 'exponent' and the result is placed in 'result'. 'base' and 'exponent' can be the same pointer and 'result' can be the same pointer as 'base' or 'exponent'. If the 'result' isn't within the range of a standard ATARI floating point number, -1 is returned as status. Otherwise 0 is returned.

FUNCTIONS USED

c_fmul
c_log10
c_alog10

EXAMPLE

```
char *pntr, bas[6], exp[6], result[6], answer[17];
pntr = "2.37";
c_afp (pntr, bas);
pntr = "7.95";
c_afp (pntr, exp);
c_exp (bas, exp, result);
c_fasc (result, answer);
printf ("%s", answer);
953.34337
```

FLOATING POINT SQUARE ROOT

PURPOSE

To take the square root of a standard ATARI floating number.

FUNCTION CALL

```
status = c_sqrt (nbr, sqroot);
```

INPUT PARAMETERS

nbr char array
pointer to a six byte character array that contains the standard ATARI floating point number whose square root is desired.

OUTPUT PARAMETERS

sqroot char array
pointer to a six byte character array that will contain the square root of '*nbr' in standard ATARI floating point format.

status integer scalar
return status:
0 = square root taken successfully.
-1 = out of range
-2 = '*nbr' is negative.

DESCRIPTION

This function takes the square root of the positive number at 'nbr' and stores it at 'sqroot'. If the square root is taken successfully, 0 is returned as status. If '*nbr' is negative, -2 is returned. If the result is out of the range of a standard ATARI floating point number, -1 is returned.

FUNCTIONS USED

c_fmul
c_log10
c_alog10

EXAMPLE

```
char *pntr, nbr[6], sqrt[6], answer[17];
pntr = "256.512";
c_afp (pntr, nbr);
c_sqrt (nbr, sqrt);
c_fasc (sqrt, answer);
printf ("%s", answer);
16.01599201
```

RETURN INTEGER PORTION OF A NUMBER

PURPOSE

To return the integer portion of a standard ATARI floating point number. The result is a floating point number.

FUNCTION CALL

```
status = c_int (nbr, intpor);
```

INPUT PARAMETERS

nbr char array

pointer to a 6 byte character array containing a standard ATARI floating point number for which the integer portion is desired.

OUTPUT PARAMETERS

intpor char array

pointer to a 6 byte character array to receive the integer portion of '*nbr'. The result is itself a standard ATARI floating point number.

status integer scalar

return status:

0 = normal completion.

-1 = no fractional portion to truncate: '*intpor' is set to '*nbr'.

-2 = '*nbr' < 1: '*intpor' set to standard ATARI floating point zero.

DESCRIPTION

The fractional part of '*nbr' is truncated and the result is stored in '*intpor'. Non-zero 'status' doesn't indicate an error condition: merely special cases, as specified above. 'nbr' and 'intpor' can be the same variable.

FUNCTIONS USED

move (in Deep Blue C AIO.CCC library)

EXAMPLE

```
char *pntr, nbr[6], intp[6], answer[17];
pntr = "1234.5678";
c_afp (pntr, nbr);
c_int (nbr, intp)
c_fasc (intp, answer);
printf ("%s", answer);
1234
```

RETURN FRACTIONAL PORTION OF A NUMBER

PURPOSE

To return the fractional portion of a standard ATARI floating point number.

FUNCTIONAL CALL

```
status = c_frac (nbr, fracpor);
```

INPUT PARAMETERS

nbr char array

pointer to a six character array containing a standard ATARI floating point number for which the fractional portion is desired.

OUTPUT PARAMETERS

fracpor char array

pointer to a six character array to receive the fractional portion of '*nbr'.

status integer scalar

return status:

0 = normal completion

-1 = '*nbr < 1': no integer portion to truncate. '*fracpor' set to '*nbr'.

-2 = no fractional portion to '*nbr'. '*fracpor' set to standard ATARI floating point zero.

DESCRIPTION:

The integer portion of '*nbr' is truncated and the result is stored at 'fracpor'. Non-zero 'status' doesn't indicate an error condition: merely special cases as indicated above. 'nbr' and 'fracpor' can be the same variable.

FUNCTIONS USED

move (in Deep Blue C AIO.CCC library)
c_int
c_fsub

EXAMPLE

```
char *pntr, nbr[6], fracp[6], answer[17];
pntr = "1234.5678";
c_afp (pntr, nbr);
c_frac (nbr, fracp)
c_fasc (fracp, answer);
printf ("%s", answer);
0.5678
```

COMPARE TWO FLOATING POINT NUMBERS

PURPOSE

To compare two floating point numbers and return an indication of the relative magnitudes of the two numbers.

FUNCTION CALL

```
result = c_cmp (fpn1, fpn2);
```

INPUT PARAMETERS

fpn1 char array
pointer to a six byte character array containing the first number to be compared

fpn2 char array
pointer to a 6 byte character array containing the second number to be compared

OUTPUT PARAMETERS

result integer scalar
an indication of the comparison:
-1 = '*fpn1' is less than '*fpn2'.
0 = '*fpn1' equals '*fpn2'.
+1 = '*fpn1' is greater than '*fpn2'.

DESCRIPTION

'*fpn1' is compared to '*fpn2'. If '*fpn1' is less than '*fpn2', 'result' is set to -1. If they're equal, 'result' is set to 0. If '*fpn1' is greater than '*fpn2', 'result' is set to +1.

FUNCTIONS USED

c_fsub

EXAMPLE

```
*pntr, nbr1, nbr2,;
int status;
pntr = "-27.45";
c_afp (pntr, nbr1);
pntr = "14.55";
c_afp (pntr, nbr2);
status = c_cmp (nbr1, nbr2);
printf ("%d", status);
-1
```

GET ABSOLUTE VALUE OF NUMBER

PURPOSE

To compute the absolute value of a standard ATARI floating point number.

FUNCTION CALL

```
c_abs (fpn, absfpn);
```

INPUT PARAMETERS

fpn char array
pointer to a 6 byte character array containing the standard ATARI floating point number for which the absolute value is desired.

OUTPUT PARAMETERS

absfpn char array
pointer to a six byte character array to receive the standard ATARI floating point absolute value of '*fpn'.

DESCRIPTION

The absolute value of '*fpn' is taken and stored at 'absfpn'. 'fpn' and 'absfpn' can be the same variable.

FUNCTIONS USED

None

EXAMPLE

```
char *pntr, nbr[6], absnbr[6], answer[17];
pntr = "-15.7895"
c_afp (pntr, nbr);
c_abs (nbr, absnbr);
c_fasc (absnbr, answer);
printf ("%s", answer);
-15.7895
```

CHANGE SIGN OF FLOATING POINT NUMBER

PURPOSE

To change the sign of a standard ATARI floating point number.

FUNCTION CALL

```
c_chs (fpn, negfpn);
```

INPUT PARAMETERS

fpn char array
pointer to a six byte character array containing a standard ATARI floating point number for which a sign change is desired

OUTPUT PARAMETERS

negfpn char array
pointer to a six byte character array to receive the negation of '*fpn'.

DESCRIPTION

The sign of '*fpn' is changed and the result is stored at 'negfpn'. 'fpn' and 'negfpn' can be the same variable.

FUNCTIONS USED

None

EXAMPLE

```
char *pntr, nbr[6], output[6], answer[17];
pntr = "15.7895"
c_afp (pntr, nbr);
c_chs (nbr, output);
c_fasc (output, answer);
printf ("%s", answer);
-15.7895
```

SET FLOATING POINT NUMBER TO ZERO

PURPOSE

To obtain a standard ATARI floating point zero.

FUNCTION CALL

```
c_zero (fpn);
```

INPUT PARAMETERS

None

OUTPUT PARAMETERS

fpn char array
pointer to a six byte character array to receive
a standard ATARI floating point zero.

DESCRIPTION

A standard ATARI floating point zero is moved to 'fpn'.

FUNCTIONS USED

move (in Deep Blue C AIO.CCC library)

EXAMPLE

```
char fpn[0], answer[17];
c_zero (fpn);
c_fasc (fpn, answer);
printf ("%s", answer);
0
```

MOVE FLOATING POINT NUMBER

PURPOSE

To move a floating point number from one place to another.

FUNCTION CALL

```
c_move (fpn1, fpn2);
```

INPUT PARAMETERS

fpn1 char array
pointer to a six byte character array containing a standard ATARI floating number to be moved.

OUTPUT PARAMETERS

fpn2 char array
pointer to a six byte character array to receive '*fpn1'.

DESCRIPTION

'*fpn1' is moved to 'fpn2'.

FUNCTIONS USED

move (in Deep Blue C AIO.CCC library)

EXAMPLE

```
char *pntr, fpn1[6], fpn2[6], answer[17];
pntr = "66";
c_afp (pntr, fpn1);
c_move (fpn1, fpn2);
c_fasc (fpn2, answer);
printf ("%s", answer);
66
```

Trigonometric functions (TRIG.CCC)

This section describes all of the trigonometric functions of MATHLIB, contained in TRIG.CCC. The following is a complete list of the trigonometric functions, in the order described in the following pages:

c_itrig:	Initialize Trigonometric Functions
c_rad:	Set Radians or Degrees
c_rd:	Convert Radians to Degrees
c_dr:	Convert Degrees to Radians
c_dmsd:	Degrees, Minutes, Seconds to Decimal Degrees
c_ddms:	Decimal Degrees to Degrees, Minutes, and Seconds
c_sin:	Compute Sine of an Angle
c_cos:	Compute Cosine of an Angle
c_tan:	Compute Tangent of an Angle
c_atan:	Compute Arctangent (Angle of a Tangent)

In the specifications that follow, the term, "decimal degrees" is used. This means degrees, including fractional degrees, expressed as a floating point number. This is in contrast to an angle expressed in degrees, minutes, and seconds. For example, the decimal degrees equivalent to 30 degrees, 25 minutes, and 37 seconds are 30.42694444 decimal degrees.

INITIALIZE TRIGONOMETRIC FUNCTIONS

PURPOSE

To initialize the trigonometric functions in MATHLIB

FUNCTION CALL

```
c_itrig();
```

INPUT PARAMETERS

None

OUTPUT PARAMETERS

None

DESCRIPTION

This function initializes the constants and variables required by the trigonometric functions of MATHLIB. It must be called before calling any of the trigonometric functions of MATHLIB. Failing to do so will cause the trigonometric functions to produce incorrect results. 'c_itrig' sets MATHLIB to operate with radians rather than degrees. See the next function to set MATHLIB to operate with degrees.

FUNCTIONS USED

c_afp

SET RADIANS OR DEGREES

PURPOSE

To tell MATHLIB whether to operate with degrees or radians when performing trigonometric operations.

FUNCTION CALL

```
c_rad (flag);
```

INPUT PARAMETERS

flag	integer	scalar

flag indicating radians or degrees:
zero = degrees
nonzero = radians

OUTPUT PARAMETERS

None

DESCRIPTION

This function tells MATHLIB whether trigonometric computations are performed in radians or degrees. It may be called at any time to change the current mode. Calling 'c_itrig' sets the mode to radians.

FUNCTIONS USED

None

CONVERT RADIANS TO DEGREES

PURPOSE

To convert radians to decimal degrees.

FUNCTION CALL

```
status = c_rd (rads, degrees);
```

INPUT PARAMETERS

rads	char	array

pointer to a 6 byte character array containing a standard ATARI floating point number specifying the number of radians to convert to degrees.

OUTPUT PARAMETERS

degrees	char	array

pointer to a six byte character array to receive the number of degrees equal to '*rads' radians, in standard ATARI floating point format.

status	integer	scalar

return status:
0 = conversion performed successfully
-1 = out of range

DESCRIPTION

This function converts radians to decimal degrees. 'rads' and 'degrees' may be the same variable.

FUNCTIONS USED:

c_fdiv

EXAMPLE

```
char *pntr, radians[6], degrees[6], answer[17];
pntr = "0.78539816";
c_afp (pntr, radians);
c_rd (radians, degrees);
c_fasc (degrees, answer);
printf ("%s", answer);
45
```

CONVERT DEGREES TO RADIANS

PURPOSE

To convert decimal degrees to radians.

FUNCTION CALL

```
status = c_dr (degrees, rads);
```

INPUT PARAMETERS

degrees	char	array

pointer to a six byte character array containing a standard ATARI floating point number specifying the number of degrees to convert to radians.

OUTPUT PARAMETERS

rads	char	array

pointer to a six byte character array to receive the number of radians equal to '*degrees' degrees, in standard ATARI floating point format.

status	integer	scalar

return status:
0 = conversion performed successfully
-1 = out of range

DESCRIPTION

This function converts decimal degrees to radians. 'degrees' and 'rads' may be the same variable.

FUNCTIONS USED

c_fmul

EXAMPLE

```
char *pntr, radians[6], degrees[6], answer[17];
pntr = "45";
c_afp (pntr, degrees);
c_dr (degrees, radians);
c_fasc (radians, answer);
printf ("%s", answer);
0.78539816
```

DEGREES, MINUTES, SECONDS TO DECIMAL DEGREES

PURPOSE

To convert degrees, minutes, and seconds to decimal degrees.

FUNCTION CALL

```
status = c_dmsd (degrees, minutes, seconds, dd);
```

INPUT PARAMETERS

degrees char array
pointer to a 6 byte character array containing a standard ATARI floating point number expressing degrees.

minutes char array
pointer to a 6 byte character array containing a standard ATARI floating point number expressing minutes.

seconds char array
pointer to a 6 byte character array containing a standard ATARI floating point number expressing seconds.

OUTPUT PARAMETERS

dd char array
pointer to a six byte character array to receive a standard ATARI floating point number that will be the decimal equivalent of '%degrees', '%minutes', and '%seconds'.

status integer scalar
return status:
0 = angle converted successfully
-1 = out of range

DESCRIPTION

An angle expressed in degrees, minutes, and seconds is converted to decimal degrees.

FUNCTIONS USED

c_fdiv
c_fadd

EXAMPLE

```
char deg[6], min[6], sec[6], ddeg[6], *aux, output[17];
aux = "30";
c_afp (aux, deg);
aux = "25";
c_afp (aux, min);
aux = "37";
c_afp (aux, sec);
c_dmsd (deg, min, sec, ddeg);
c_fasc (ddeg, output);
printf ("%s", output);
30.42694444
```

DECIMAL DEGREES TO DEGREES, MINUTES, AND SECONDS

PURPOSE

To convert decimal degrees to degrees, minutes and seconds.

FUNCTION CALL

```
status = c_ddms (dd, degrees, minutes, seconds);
```

INPUT PARAMETERS

dd	char	array

pointer to a 6 byte character array containing a standard ATARI floating point number representing the decimal degrees to be converted

OUTPUT PARAMETERS

degrees	char	array

pointer to a 6 byte character array to receive a standard ATARI floating point number expressing degrees.

minutes	char	array

pointer to a 6 byte character array to receive a standard ATARI floating point number expressing minutes.

seconds	char	array

pointer to a 6 byte character array to receive a standard ATARI floating point number expressing seconds.

status	integer	scalar

return status:
0 = angle converted successfully
-1 = out of range

DESCRIPTION

An angle expressed in decimal degrees is converted to an angle expressed in degrees, minutes, and seconds.

FUNCTIONS USED

c_int
c_fsub
c_fmul

EXAMPLE

```
char deg[6], min[6], sec[6], ddeg[6], *aux;
char out1[17], out2[17], out3[17];
aux = "30.42694444";
c_afp (aux, ddeg);
c_ddms (ddeg, deg, min, sec);
c_fasc (deg, out1);
c_fasc (min, out2);
c_fasc (sec, out3);
printf ("%s, %s, %s", out1, out2, out3);
```

30, 25, 37

COMPUTE SINE OF AN ANGLE

PURPOSE

To compute the sine of an angle

FUNCTION CALL

```
status = c_sin (angle, sine);
```

INPUT PARAMETERS

angle char array
Pointer to a six byte character array containing a standard ATARI floating point number which is the decimal angle for which the sine is desired.

OUTPUT PARAMETERS

sine Pointer to a six byte character array to receive the sine of '*angle' in standard ATARI floating point format.

status integer scalar
return status:
0 = sine computed correctly.
-1 = out of range

DESCRIPTION

The sine of '*angle' is computed and stored at 'sine'. The angle is reduced to the range 0 <= 'angle' <= +pi/4 and eight terms of the Taylor Series are used to compute the sine to eight digits of accuracy.

FUNCTIONS USED

move (from Deep Blue C AIO.CCC library)
C_fmul
c_fdiv
c_frac
c_fsub
C_fadd

EXAMPLE

```
char *pntr, nbr[6], sinnbr[6], answer[17];
rad(0); /* set degrees */
pntr = "30";
c_afp (pntr, nbr);
c_sin (nbr, sinnbr);
c_fasc (sinnbr, answer);
printf ("%s", answer);
0.5
```

COMPUTE COSINE OF AN ANGLE

PURPOSE

To compute the cosine of an angle

FUNCTION CALL

```
status = c_cos (angle, cosine);
```

INPUT PARAMETERS

angle char array
Pointer to a 6 byte character array containing a standard ATARI floating point number that is the decimal angle for which the cosine is desired.

OUTPUT PARAMETERS

cosine Pointer to a 6 byte character array to receive the cosine of '*angle' in standard ATARI floating point format.

status integer scalar
return status:
0 = cosine computed correctly.
-1 = out of range

DESCRIPTION

The cosine of '*angle' is computed and stored at 'cosine'.

FUNCTIONS USED

move (from Deep Blue C AIO.CCC library)
c_fmul
c_fsub
c_sin

EXAMPLE

```
char *pntr, nbr[6], cosnbr[6], answer[17];
rad(0); /* set degrees */
pntr = "30";
c_afp (pntr, nbr);
c_cos (nbr, cosnbr);
c_fasc (cosnbr, answer);
printf ("%s", answer);
0.8660254
```

COMPUTE TANGENT OF AN ANGLE

PURPOSE

To compute the tangent of an angle

FUNCTION CALL

```
status = c_tan (angle, tangent);
```

INPUT PARAMETERS

angle char array
Pointer to a 6 byte character array containing a standard ATARI floating point number that is the decimal angle for which the tangent is desired.

OUTPUT PARAMETERS

tangent Pointer to a 6 byte character array to receive the tangent of '*angle' in standard ATARI floating point format.

status integer scalar
return status:
0 = tangent computed correctly.
-1 = out of range

DESCRIPTION

The tangent of '*angle' is computed and stored at 'tangent'.

FUNCTIONS USED

c_sin
c_cos
c_fdiv

EXAMPLE

```
char *pntr, nbr[6], tannbr[6], answer[17];
rad(0); /* set degrees */
pntr = "30";
c_afp (pntr, nbr);
c_tan (nbr, tannbr);
c_fasc (tannbr, answer);
printf ("%s", answer);
```

0.57735027

COMPUTE ARCTANGENT

PURPOSE

To compute the arctangent of a floating point number.

FUNCTION CALL

status = c_atan (tangent, angle);

INPUT PARAMETERS

tangent char array
pointer to a 6 byte character array that contains a standard ATARI floating point number for which the arctangent is desired.

OUTPUT PARAMETERS

angle pointer to a six byte character array that will contain the arctangent of '*tangent' in standard ATARI floating point format

status integer scalar
return status:
0 = arctangent correctly computed
-1 = out of range

DESCRIPTION

The arctangent of '*tangent' is taken and stored at 'angle'. A high quality 10 term polynomial evaluation is used to compute the arctangent to 9 1/2 digits of accuracy. The result will range -90 < 'angle' < +90 in degrees or -pi/2 < 'angle' < +pi/2 in radians (depending on the current trig mode of MATHLIB).

FUNCTIONS USED

move (from Deep Blue C AIO.CCC library)
c_fdiv
c_cmp
c_fmul
c_fadd
c_fsub

EXAMPLE

```
char pntr, deg45[6], atan45[6], answer[17];
rad (0); /* set mode to degrees */
pntr = "1";
c_afp (pntr, deg45);
c_atan (deg45, atan45);
c_fasc (atan45, answer);
printf ("%s", answer);
45
```

APX ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

Review Form

We're interested in your experiences with APX programs and documentation, both favorable and unfavorable. Many of our authors are eager to improve their programs if they know what you want. And, of course, we want to know about any bugs that slipped by us, so that the author can fix them. We also want to know whether our instructions are meeting your needs. You are our best source for suggesting improvements! Please help us by taking a moment to fill in this review sheet. Fold the sheet in thirds and seal it so that the address on the bottom of the back becomes the envelope front. Thank you for helping us!

1. Name and APX number of program.

   Mathlib (231)

2. If you have problems using the program, please describe them here.

3. What do you especially like about this program?

4. What do you think the program's weaknesses are?

5. How can the catalog description be more accurate or comprehensive?

6. On a scale of 1 to 10, 1 being "poor" and 10 being "excellent", please rate the following aspects of this program:

   _____ Easy to use
   _____ User-oriented (e.g., menus, prompts, clear language)
   _____ Enjoyable
   _____ Self-instructive
   _____ Use (non-game programs)
   _____ Imaginative graphics and sound

7. Describe any technical errors you found in the user instructions (please give page numbers).

8. What did you especially like about the user instructions?

9. What revisions or additions would improve these instructions?

10. On a scale of 1 to 10, 1 representing "poor" and 10 representing "excellent", how would you rate the user instructions and why?

11. Other comments about the program or user instructions:

From

STAMP

APX

ATARI Program Exchange
P.O. Box 3705
Santa Clara, CA 95055

[seal here]